Rapport projet compilateur - VLOBJ

Étudiants : Izzo Valentino et Frossard Loïc
Professeur : François Tièche

1. Introduction

Dans le cadre du cours compilateur, un projet doit être réalisé regroupant les connaissant apprises durant le semestre. Le nom du projet est VLOBJ. C'est un compilateur qui doit convertir du vlobj qui est un langage créer pour le projet, en deux fichiers *.obj* et *.mtl* qui sont lisible par des logiciels de modélisation 3D comme Blender. Le but du vlobj est de pouvoir créer, positionner, appliquer une taille et une couleur à des cubes ou pyramides.

2. Analyse

Ce chapitre décrit en détail toutes les phases d'analyse de ce projet.

2.1 VLOBJ

Voici, les différentes possibilités de syntaxe qu'offre notre langage qui est le VLOBJ.

2.1.1 L'assignation

L'assignation de variable est possible avec le symbole **"=)"**. Une variable accepte que des nombres et non des chaînes de caractères.

```
x =) 2
```

2.1.2 Incrémentation / Décrémentation

Il est possible d'incrémenter ou décrémenter un nombre dans une variable de 1. Le symbole pour ajouter est **"+_+"** et pour réduire **"-_-"**.

> Attention les commentaires "//" ne fonctionne pas dans le langage vlobj

```
x =) 0;
x+_+; // x vaut 1
x-_- // x vaut 0
```

2.1.3 Les formes

Le vlobj permet de créer deux formes, le cube et la pyramide. Les mots clefs sont **"Square(arguments)"** et **"Pyramid(arguments)"**. Voici les arguments, la position (x,y,z), la taille et le nom d'une couleur. Si le dernier argument n'est pas renseigné, la forme n'aura pas de couleur.

```
Square(1,1,1,2,'bleu'); //Cube positionner en (1,1,1) de taille 2 et de couleur
bleu
Pyramid(5,5,5,1) //Pyramide positionner en (5,5,5) de taille 1 et sans couleur
```

Il est possible aussi de créer une forme aléatoirement. Avec le symbole **"xS(arguments)"**, une forme aléatoire entre le cube et la pyramide est créée. Les arguments restent les mêmes que précédemment.

```
xS(2, 4, 5, 2, 'red')
```

2.1.4 Couleur

Pour colorier les formes créées, utiliser la fonction **"Color(nom, r, g, b)"**.

```
Color('bleu', 0, 0, 255)
```

2.1.4 Les tests

Il est possible d'effectuer un test et donc de réaliser une partie de code ou une autre. Le symbole équivalent aux "if" est **":/"** et le **"else"** reste le même. Les conditions possibles sont **"=="**, **">"** et **"<"**. Les accolades sont représentées par les symboles **":("** (ouvrante) et **"):"** (fermante).

```
:/(2 == 2):(
   xS(2,2,2,5)
):else:(
  Square(1,1,1,2,'bleu')
:)
```

Un test peut avoir dans sa condition des variables.

```
x =) 1
:/(2 > x):(
   xS(2,2,2,5)
):else:(
  Square(1,1,1,2,'bleu')
:)
```

2.1.5 Les boucles

Les boucles sont réalisables avec le symbole **"^^"**. La condition de la boucle peut être comme pour les tests soit "==", "<" ou ">". Les accolades sont aussi les mêmes que les tests. La boucle s'arrête quand la condition est validée.

```
i =) 0;
^^(i < 10) :(
    Square(i,1,i,2,'bleu');
    i+_+
):
```

2.1.6 Nombre aléatoire

Le vlobj contient aussi une fonctionnalité comme le "random" en python. Avec la fonction **"xD(min, max)"**, un nombre aléatoire est généré entre le premier argument(min) et le deuxième(max). Le nombre aléatoire peut être directement assigné à une variable pour être utilisé dans le code.

```
nb =) xD(1,10);
Square(nb,2,2,5,'green')
```

2.1.7 Gestion du point-virgule

Le point-virgule est très particulier, il sépare deux parties de code. Le point-virgule va se mettre après chaque ligne sauf à la fin d'un bloc. Se que nous appelons bloc est soit l'entier du fichier ou entre les symboles **:(** et **):**.

```
Color('vert', 0, 255, 0);

i =) 0;
^^(i < 10000) :(
    x =) xD(-200,200);
    y =) xD(-200,200);
    z =) xD(-200,200);
    n =) xD(1,10);

    :/(x < 0) :(
        xS(x,y,z,n,'vert') //pas ici, car a la fin du bloc (if)
    ):;
    i+_+ //pas ici, car a la fin du bloc (while)
): //pas ici, car a la fin du bloc (fichier)
```

2.2 Fichiers .obj et .mtl

Avant de commencer le projet, il a fallu ce documenter sur comment fonctionne les fichiers *.obj* et *.mtl*. Car la sortie du compilateur renvoie deux fichiers de ce type à ouvrir dans le logiciel de modélisation 3D qui permet de visualiser les cubes et pyramides codées en vlobj.

2.2.1 OBJ

Le format obj permet de décrire la géométrie 3D d'objet. Le fichier se compose de plusieurs parties.

1. Le nom du fichier *.mtl* qui est utilisé par le *.obj*

```
mtllib input1.mtl
```

3. Le nom de l'objet, dans le cas du projet soit "Cube" ou "Pyramid". Le nom n'a pas réellement d'importance, mais il est nécessaire dès qu'il y a plusieurs objets, car sinon c'est impossible de les différencier.

```
o Cube
o Pyramid
```

3. Les sommets que compose la forme avec leurs coordonnées (x,z,y). Ce n'est pas une erreur, le *z* vient avant *y*. Selon certaines documentations, la position est (x,y,z), mais suite à nos tests et analyses nous avons des résultats corrects avec l'ordre suivant (x,z,y).

```
v 1.0 0.0 0.0
```

4. Les coordonnées de texture

```
vt 1.0 0.0
```

5. Les normales des faces

```
vn 0.0 1.0 0.0
```

6. Pour terminé, la suite des indices des faces

```
f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3
```

2.2.2 MTL

Le format MTL est un standard qui est un complément au format OBJ. Le fichier contient les différents matériaux des objets. Dans le projet, c'est utilisé pour sauvegarder les couleurs. Le fichier se compose de plusieurs parties.

1. Nom du matériel

```
newmtl bleu
```

2. Le specular exponent entre 0 et 100

```
Ns 96.078431
```

3. La couleur en RGB.

```
Ka 1.000000 1.000000 1.000000
```

4. La couleur diffuse

```
Kd 0.640000 0.640000 0.640000
```

5. La couleur spéculaire

```
Ks 0.500000 0.500000 0.500000
```

6. La couleur émissive

```
Ke 0.000000 0.000000 0.000000
```

7. La densité optique

```
Ni 1.000000
```

8. La transparence

```
d 1.000000
```

9. Les paramètres de lumières

```
illum 2
```

Après analyse du fichier, le projet utilise seulement **Kd** pour la couleur RGB. Les autres paramètres restent par défaut.

3. Réalisation

Ce chapitre explique en détail la réalisation du projet.

3.1 Analyse lexical

L'analyse lexicale est effectuée avec le fichier `lexVLOBJ.py`. Cette analyse permet de détecter les symboles du langage VLOBJ décrit dans le chapitre *Analyse*.

Un problème est survenu lors de la gestion des symboles. Les mettre dans la liste des mots réservés faisait buger l'application, car le lexeur ne reconnaissait pas les symboles du coup on a du les mettre sous la forme de regex.

3.2 Analyses syntaxiques

L'analyse syntaxique correspond à l'obtention de relation entre les symboles qui forment le vlobj.

3.2.1 IF

Le noeud du if se créer avec un enfant gauche qui sera la condition et un enfant droit qui est le bloc intérieur du if

3.2.2 Condition

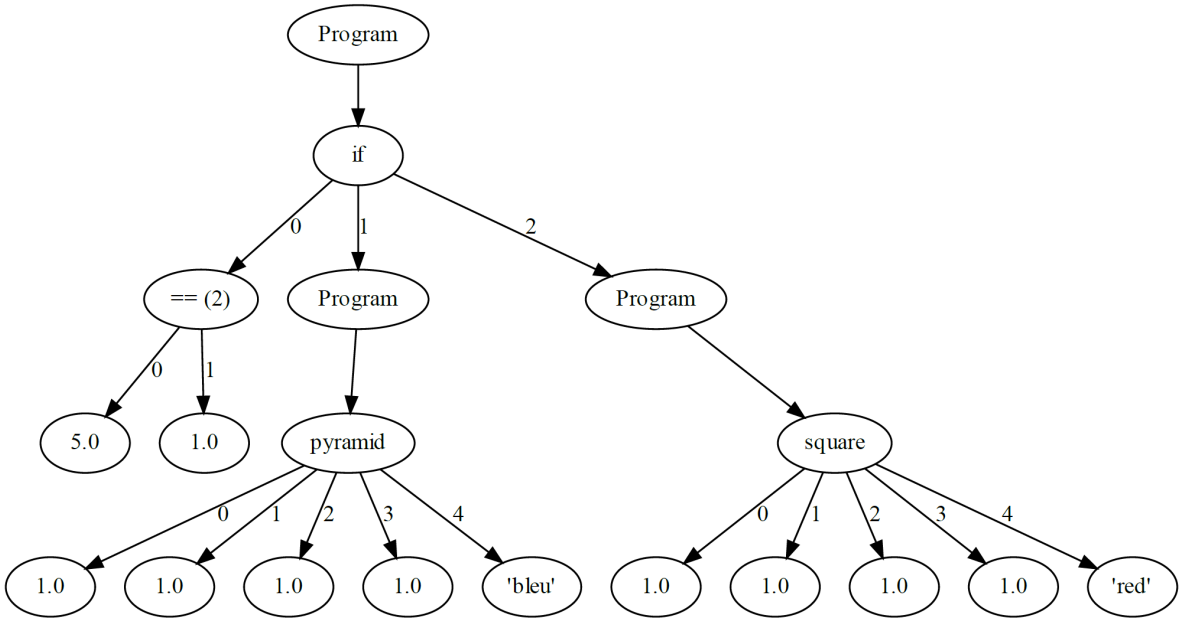
La condition se créer avec un paramètre qui est la condition. Deux enfants qui sont les deux chiffres a tester.

3.2.3 Shape (Square, Pyramid, Random Shape)

Les trois premiers enfants sont pour la position x, y et z. Ensuite la taille et la couleur s'il y en a une.

3.2.4 Arbre syntaxique

Le parseur génère à la fin, un arbre syntaxique qui permet de visualiser les liaisons entre chaque noeud avec le parent et les enfants. Ce fichier *.pdf* facilite la compréhension du parseur et la correction de problème. Voici, un exemple qui contient deux formes et un "if else".



3.3 Compilateur

Le but du compilateur sera de générer deux fichiers, le **.obj** et le **.mtl**. Il va s'aider du fichier `constanteVLOBJ.py` pour calculer les constantes du cube, de la pyramide et de la couleur. Certaines valeurs des fichiers ne changent en rien le rendu final de la forme, c'est pourquoi elle se retrouve en constante dans le fichier.

3.3.1 Shape

Les quatre premiers enfants sont testés pour avoir une suite de 4 float, si un de ces quatre enfants est une string ou contient une erreur la forme ne s'intègre pas dans le **.obj** et l'utilisateur est notifié par une ligne d'erreur qui s'affiche dans la console.

S'il se trouve un 5e enfant, une couleur sera attribuée a la shape et elle sera ajouté dans le fichier **.mtl** sinon elle aura la couleur par défaut dans le logiciel de rendu 3D.

3.3.2 Assignation / Token

Un tableau associatif permet de sauvegarder les assignations effectuées et les réutiliser dans d'autres fonctions. Quand un noeud *assignation* est détecté, cela va ajouter dans la clef, le nom de la variable et la valeur correspond à l'élément situé après le symbole "=)".

Quand le token est appelé, si c'est une string. Il retourne le nombre qui est associé à cette suite de caractère dans le tableau associatif.

3.3.3 Condition

La condition contient un paramètre d'opération qui peut être **<**, **>** ou **==** et deux enfants. Grâce à un tableau de fonction, on retourne le résultat de l'opération.

```
conditions = {
    '==': lambda x, y: x==y,
    '<': lambda x, y: x < y,
    '>': lambda x, y: x > y,
}
```

3.3.4 Test et while

Pour les tests et boucles, le compilateur va continuer la compilation du programme contenu dans le test ou la boucle seulement si la condition est validée.

3.3.5 Random Shape

Tout d'abord on va choisir un booléen aléatoire pour savoir si le programme va faire une *Pyramide* ou un *Cube*. Ensuite la fonction créer un nouveau node et retourne le résultat de la compilation.

4. Résultats obtenus

Voici, trois résultats obtenus. Le premier un test simple qui permet de contrôler le fonctionnement des formes. Le deuxième montre la détection d'erreur. Le dernier est un test avec toutes les fonctionnalités du vlobj. Bien évidemment, nous avons réalisé des tests pour chaque fonctionnalité séparée en petites parties qui sont dans le projet. Cependant cela prend beaucoup de place si chaque test est monté dans le rapport. Chaque test ne présente aucun problème seulement le fichier "inpu7.vlobj" qui est un test contenant des erreurs voulues pour vérifier leurs captations.

4.1 Les formes

Voici, un exemple très simple avec une forme qui est un cube de position (1, 2, 3) et de taille 4 avec une couleur bleue et une pyramide de position (10, 0, 0) et de taille 2 avec aucune couleur.

```
Color('bleu', 0, 0, 255);
Square(1, 2, 3, 4,'bleu');
Pyramid(10, 0, 0, 2)
```

Dans le fichier *.obj*, il y a bien les deux formes, le cube est bien associé avec la couleur bleue et la pyramide associée à aucun matériel (None).

```
mtllib input1.mtl
o Cube
v -1.0 1.0 -0.0
v -1.0 5.0 -0.0
v -1.0 1.0 -4.0
v -1.0 5.0 -4.0
v 3.0 1.0 -0.0
v 3.0 5.0 -0.0
v 3.0 1.0 -4.0
v 3.0 5.0 -4.0
vt 0.625000 0.500000
vt 0.875000 0.500000
vt 0.875000 0.750000
vt 0.625000 0.750000
vt 0.375000 0.750000
vt 0.625000 1.000000
vt 0.375000 1.000000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.625000 0.250000
vt 0.375000 0.250000
vt 0.125000 0.500000
```

```
vt 0.375000 0.500000
vt 0.125000 0.750000
vn 0.0000 1.0000 0.0000
vn 0.0000 0.0000 1.0000
vn -1.0000 0.0000 0.0000
vn 0.0000 -1.0000 0.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
usemtl bleu
s off
f 1/1/1 2/2/1 4/3/1 3/4/1
f 3/4/2 4/3/2 8/5/2 7/6/2
f 7/6/3 8/5/3 6/7/3 5/8/3
f 5/8/4 6/7/4 2/9/4 1/10/4
f 3/11/5 7/6/5 5/8/5 1/12/5
f 8/5/6 4/13/6 2/14/6 6/7/6
o Pyramid
v 10.0 -0.25 0.65
v 10.5 -0.25 -0.35
v 9.5 -0.25 -0.35
v 10.0 0.75 0.05
vt 0.250000 0.490000
vt 0.250000 0.250000
vt 0.457846 0.130000
vt 0.750000 0.490000
vt 0.957846 0.130000
vt 0.542154 0.130000
vt 0.042154 0.130000
vn 0.8639 0.2592 -0.4319
vn 0.0000 -1.0000 0.0000
vn -0.0000 0.3714 0.9285
vn -0.8639 0.2592 -0.4319
usemtl None
s off
f 9/15/7 12/16/7 10/17/7
f 9/18/8 10/19/8 11/20/8
f 10/17/9 12/16/9 11/21/9
f 11/21/10 12/16/10 9/15/10
```

```
newmtl bleu
Ns 225.000000
Ka 1.000000 1.000000 1.000000
Kd 0.0 0.0 1.0
Ks 0.500000 0.500000 0.500000
Ke 0.000000 0.000000 0.000000
Ni 1.450000
d 1.000000
illum 2
```

4.2 Erreurs

Voici, un test des erreurs de syntaxe que le compilateur peut détecter. La première erreur est que "x" est assigner à une *string* cependant ce n'est pas un comportement voulu. La prochaine erreur prévient qu'il est impossible d'appliquer des *string* à des arguments de positions qui doivent être des *int*.

```
x =) 'aaa';
Pyramid(10,x,3,5,'bleu')
```

Ci-dessous, les messages d'erreurs que le compilateur affiche pour prévenir l'utilisateur.

```
ERROR : Assignation must be integer !!
ERROR : An attribute of Pyramid expect to be an int but it's a string !!
```

4.3 10'000 formes

Un *stress* test a été effectué en créant 10'000 objets via une boucle. Les formes ont été générées aléatoirement, leur taille et position aussi. En fonction de la position de la forme, une couleur différente est appliquée.

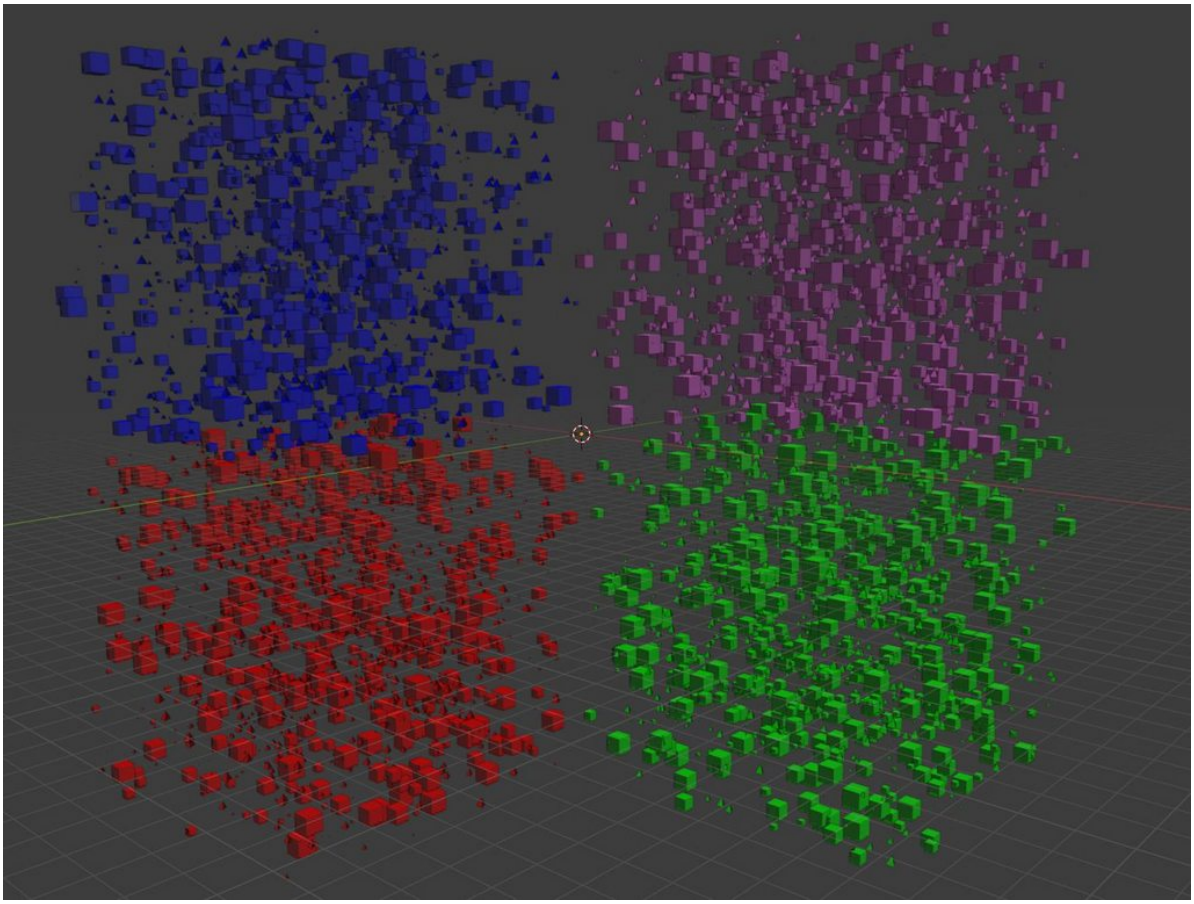
```
Color('bleu', 0, 0, 255);
Color('rouge', 255, 0, 0);
Color('vert', 0, 255, 0);
Color('rose',235, 52, 210);

i =) 0;
^^(i < 10000) :(
    x =) xD(-200,200);
    y =) xD(-200,200);
    z =) xD(-200,200);
    n =) xD(1,10);

    :/(x < 0) :(
        :/(y < 0) :(
            :/(z < 0) :(
                xS(x,y,z,n,'vert')
            ):else:(
                xS(x,y,z,n,'rose')
            ):
        ):
    ):;
    :/(x > 0) :(
        :/(y > 0) :(
            :/(z > 0) :(
                xS(x,y,z,n,'bleu')
            ):else:(
                xS(x,y,z,n,'rouge')
            ):
        ):
    ):;
    i+_+
):
```

Les fichiers de sortie du compilateur sont beaucoup trop grands pour être montré dans le rapport.

Après quelques minutes de compilation, les fichiers *.obj* et *.mtl* ont été créés. Ensuite, l'importation des fichiers dans Blender prend aussi plusieurs minutes de chargements et voici le résultat ci-dessous. On peut voir la séparation des formes via les différentes couleurs.



Cet exemple final montre bien les possibilités du vlobj et de la réussite du compilateur.

5. Conclusion

En conclusion de ce projet du cours compilateur, grâce à une bonne cohésion du groupe et de notre analyse de projet, nous sommes arrivés à un projet respectant le cahier des charges et les contraintes de la donnée.